Bringing PlantUML to Kieker: Streamline the Visualization Pipeline Toward Kieker–Jupyter Notebook Interoperability

Yorrick Josuttis
Kiel University

Shinhyung Yang
Kiel University

Wilhelm Hasselbring
Kiel University

Abstract

Visual analysis is important to understand the target program architecture. Kieker's Trace Analysis tool allows visual analysis of a target program by reading traces obtained by Kieker's agent-based dynamic analysis. Our goal to provide Trace Analysis on Jupyter Notebook is hindered by the current processing pipeline, which requires additional external processing steps to generate visualization files. They are prone to system-level errors such as scripting and dependency updates. Moreover, Kieker's current sequence diagram breaks if there are too many message callers/receivers. With our new PlantUML-based approach, we obviate external process calls and incorporate them into Kieker's pipe-and-filter architecture. The PlantUML library allows processing the DOT diagram file type, and does not break the visualization even if the content gets bigger. We also added aggregation of diagrams, e.g., sequence diagrams, into a multi-page PDF, if there are more than one of that diagram type.

1 Introduction

Observability data are not easy to interpret directly. Kieker reconstructs traces and architectural models from monitoring data and visualizes them, e.g., as sequence diagrams, call trees, or dependency graphs. Those visualizations help to understand the run-time behavior and the structure of a system. For this, Kieker needs a reliable rendering pipeline.

The existing implementation needs multiple external rendering steps: (1) using the *GNU plotutils* commands to convert *PIC* into *PostScript*, (2) using the *Ghostscript* interpreter for further conversion into PDF or using *Graphviz* to convert DOT into PDF. In addition to the external factor, the rendering process is inconsistent for PNG, SVG, and PDF files. Thus, the visualization workflow is fragmented and the implementation is harder to maintain.

As part of the SustainKieker project,[1] our objective is to provide UI for Trace Analysis functionality using Jupyter Notebooks.[2] Our current Notebook example [3] relies on platform-dependant external programs as intermediate rendering steps. They are not intuitive and prone to user mistakes.

We introduce a *PlantUML* pipeline for *Kieker*'s Trace Analysis. This pipeline uses the *TeeTime* pipe-and-filter (P&F) framework. It supports the generation of PNG, SVG, and PDF graphic types for sequence diagrams, call trees, and dependency graphs.

Using the PlantUML library, we are capable of rendering diagrams inside Kieker's Trace Analysis process. By using a generic TeeTime stage for converting PlantUML files into graphic types, everything is rendered in a single place.

2 Background

2.1 Kieker Observability Framework

Kieker is a performance monitoring and dynamic software analysis framework [2, 4, 6]. It has the capability of online and offline analysis, i.e., analyzing previously collected or incoming monitoring records.

The Trace Analysis tool supports analyzing and visualizing method calls and their associated execution times. This tool allows us to reconstruct the architectural models of the monitored systems and visualize them in diagrams.[4]

2.2 TeeTime P&F Framework

TeeTime is a generic P&F framework used by Kieker to implement analysis pipelines [3].

It is capable of modeling and running P&F architectures in parallel with high performance and low overhead. Each computation component is represented as a *stage* with typed input and output ports. Pipes connect these ports and transmit data between stages. A configuration defines the stages and their connections that model a whole pipeline.

3 Modernizing the Diagram & Graph Generation

3.1 Current Approach: External Pipeline

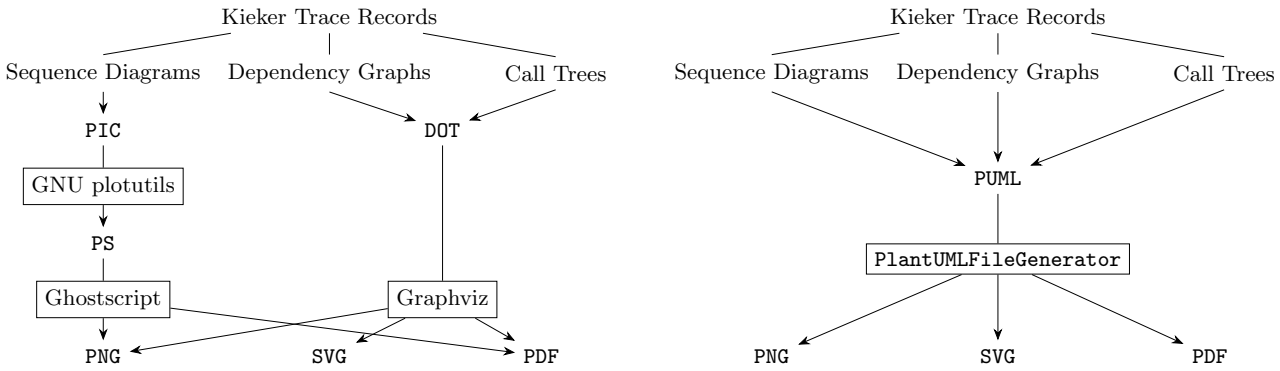
The existing pipeline uses PIC for the visualization of sequence diagrams and DOT for the visualization of

[1] `https://sustainkieker.kieker-monitoring.net/`, funded by the DFG, grant no. 528713834

[2] `https://jupyter.org/`

[3] `https://github.com/YorrickJosuttis/KiekerTraceAnalysisPlantumlDemo`

[4] `https://kieker-monitoring.readthedocs.io/en/latest/tutorials/How-to-perform-Trace-Analysis.html`



(a) Legacy visualization pipeline with external tools.

(b) PlantUML-based visualization pipeline.

Figure 1: Comparison of both visualization pipelines, where in (a) is the existing implementation using PIC/DOT with external tools, and in (b) is our PlantUML-based approach using a unified file type `PUML`.

dependency graphs and call trees. PIC is a language for drawing pictures introduced in 1982, where the basic elements are boxes, lines, arrows, circles, ellipses, arcs, and splines [1]. In Figure 1(a), we explain how the current pipeline works, which depends on external tools for file conversion. As we can see, external tools are needed to transform PIC files into some desired graphic types. Therefore, GNU plotutils[5] transforms PIC files into PostScript files, and Ghostscript[6] transforms those PostScript files further into graphic files. For the creation of sequence diagrams, Kieker uses Spinellis macros.[7]

Graphviz is open-source software to visualize graphs.[8] It has their own domain-specific language (DSL), DOT, which is an abstract grammar for defining Graphviz elements. Here too, as illustrated in Figure 1(a), an external tool, namely Graphviz, is used to convert DOT files into the desired graphic files. Kieker uses DOT for the creation of dependency graphs and call trees.

3.2 New Approach: Internal Pipeline

PlantUML is an open-source software for creating (non-)UML diagrams from textual descriptions.[9][10] It provides its own DSL and supports various output formats, including PNG and SVG. It uses Graphviz for rendering graphs described by DOT.

As we can see in Figure 1(b), for the new PlantUML pipeline, we defined a new TeeTime configuration. For each diagram type, we implement a dedicated TeeTime stage. For the creation of dependency graphs and call trees, we preserve the existing DOT generation while integrating it into a uniform PlantUML rendering pipeline. For that, we wrapped the DOT description into `@startdot` and `@enddot` syntax. Hence, all diagram type descriptions can be stored in the PUML format. The core of the pipeline includes a `PlantUMLFileGenerator` class, which is realized as an `AbstractConsumerStage` of the TeeTime framework. It consumes the generated PUML files and is capable of creating PNG, SVG, or PDF files of its content. The rendering of PNG and SVG is provided directly through the PlantUML Java library. PDF generation additionally relies on Apache Batik[11] and Apache FOP.[12] The `PlantUMLFileGenerator` merges PDF files into one PDF file using Apache PDFBox,[13] if multiple PUML files are provided.

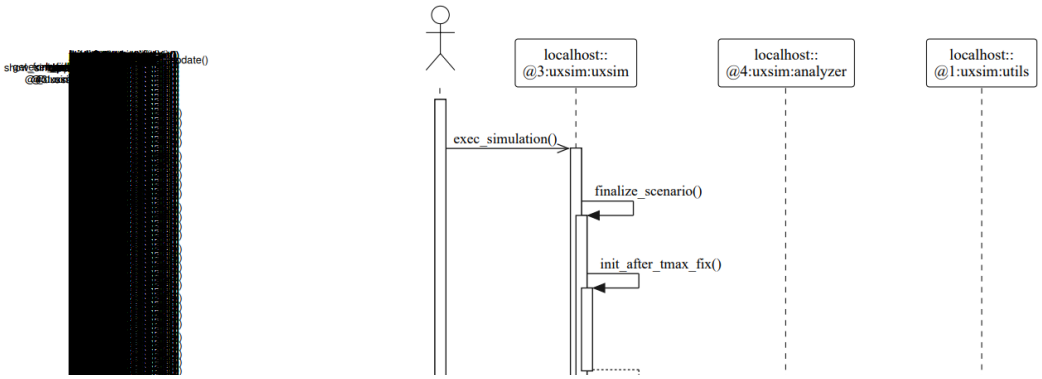
With the legacy PIC approach, the output graphic may contain a diagram, which is squeezed together, and parts overlap as in Figure 2(a). Thus, those diagrams are unreadable. Note that this figure shows only a small excerpt of the diagram. The PlantUML pipeline addresses this problem, allowing us to generate readable diagrams as in Figure 2(b).

With the new pipeline, the rendering process is invoked and managed within the Kieker Trace Analysis program instead of requiring users to execute separate conversion commands manually. I.e., Kieker Trace Analysis creates graphics files from Kieker records invoked within its Java implementation. For each diagram type, we implemented a diagram-specific TeeTime stage that is integrated into the existing TeeTime-based Trace Analysis implementation. All diagram descriptions are stored in a common PUML format. Thus, we use the `PlantUMLFileGenerator` as the same output infrastructure for all formats. The merging of multiple PDF files occurs within the pipeline.

3.3 Validation

We validated the PlantUML pipeline using trace data obtained from an example Python software UXsim [5]. Our validation consists of two stages. (1) processing

[5] `https://www.gnu.org/software/plotutils/`
[6] `https://ghostscript.com/`
[7] `https://github.com/dspinellis/UMLGraph/`
[8] `https://graphviz.org/`
[9] `https://plantuml.com/`
[10] License Compatibility: To ensure Kieker's Apache Software License (ASL), we opt for ASL build of PlantUML Java library.
[11] `https://xmlgraphics.apache.org/batik/`
[12] `https://xmlgraphics.apache.org/fop/`
[13] `https://pdfbox.apache.org/`



(a) Legacy PIC-based output (b) PlantUML-based output

Figure 2: Example sequence diagram output comparison. (a) and (b) draws the same sequence diagram but (a) do not position lifelines and call messages proportionally. Complete diagram files are available on our Zenodo repository. See Section 3.3.

Kieker traces and generating different PUML files including sequence diagram, dependency graph, and call trees. (2) Generating different visualization formats PNG, SVG, and PDF from the generated PUML file without any visual clutters. Our pipeline successfully passed all stages. The new sequence diagram in Figure 2(b) do not show any visual clutters. We made the generated data available on a permanent data repository.[14]

4 Related Work

Text-based, open-source UML rendering software includes mermaid,[15] while GUI-based, proprietary modeling includes Visual Paradigm.[16] We choose PlantUML over mermaid for more extensive UML support, and Java tooling. The Eclipse Modeling Framework[17] focuses on code generation using its Ecore model, which can be converted from and to UML.

5 Future Work

Our current work includes automated tests of the proposed pipeline and Java-Python interoperability using *Py4J*[18], to call Kieker functions and show the generated graphics direclty within Jupyter Notebook. Our ultimate goal is to make it intuitive for users to use Kieker within Jupyter Notebook, obviating external Trace Analysis executions.

6 Conclusion

We presented a new PlantUML-based TeeTime pipeline for Kieker's Trace Analysis tool. The visualization of sequence diagrams, dependency graphs, and call trees into PNG, SVG, and PDF formats are streamlined, removing external processes. We further improved the existing P&F architecture with a common PlantUML representation and a new, versatile TeeTime stage, `PlantUMLFileGenerator` stage, which renders the input visualization into a desired output format in a single stage. It replaces the PIC generation, while reusing DOT descriptions. Because Kieker is written in the object-oriented programming language Java, the effort for extending the software was straight-foward. We validated our pipeline by generating different visualizations of an example Python software and our new sequence diagram output do not exhibit overlapping of call messages and life lines.

[14] `https://doi.org/10.5281/zenodo.23258196`
[15] `https://mermaid.ai/`
[16] `https://www.visual-paradigm.com/`
[17] `https://eclipse.dev/emf/`
[18] `https://www.py4j.org/`

References

[1] B. W. Kernighan. "PIC — A language for typesetting graphics". In: *Software: Practice and Experience* 12.1 (1982), pp. 1–21. DOI: 10.1002/spe.4380120102.

[2] A. van Hoorn, J. Waller, and W. Hasselbring. "Kieker: a framework for application performance monitoring and dynamic software analysis". In: *ICPE*. 2012, pp. 247–248. DOI: 10.1145/2188286.2188326.

[3] C. Wulf, W. Hasselbring, and J. Ohlemacher. "Parallel and Generic Pipe-and-Filter Architectures with TeeTime". In: *ICSAW*. 2017. DOI: 10.1109/ICSAW.2017.20.

[4] W. Hasselbring and A. van Hoorn. "Kieker: A monitoring framework for software engineering research". In: *Software Impacts* 5 (2020). DOI: 10.1016/j.simpa.2020.100019.

[5] T. Seo. "UXsim: lightweight mesoscopic traffic flow simulator in pure Python". In: *Journal of Open Source Software* 10.106 (2025), p. 7617. DOI: 10.21105/joss.07617.

[6] S. Yang et al. "The Kieker Observability Framework Version 2". In: *ICPE*. 2025, pp. 11–15. DOI: 10.1145/3680256.3721972.